1.1 Supervised dataset

1. What are the train, val and test sets used for?

When we would like to use a model for a specific task, we usually go through several stages. The main ones are training, which is the time when you want your model to learn the task; validation, which is used to validate your training (for example, to understand if it is overfitting) or to select the best model with the best hyper-parameters; and finally, testing your model by asking it to perform the intended task. It is important that each step has its own portion of the data that is not the same as the data used in other steps. For example, if you test your model with the same data used in training, you would probably get good results, but it would be akin to cheating. The same applies to using the validation set as the test set.

In the end:
➤ Training set: used to train a given/chosen model on this data.
➤ Validation set: used for model selection (assists in fine-tuning the model). The test set should remain untouched to avoid falsifying the generalization error.
➤ Test set: used for the final model evaluation. This data should be kept separate from the training process. During the final evaluation, the model encounters this data for the first time.

2. What is the influence of the number of examples N ?

Typically, the greater the number of available examples, the better the results you can achieve. If one trains a complex model with too little data, the model parameters will overly adapt to the limited training examples, potentially leading to overfitting (⇒ see bias-variance trade-off for more information on overfitting and model choice). Usually, there's no downside to training a model with too much data, except that it would be much more time-consuming.

1.2 Network architcture (forward)

3. Why is it important to add activation functions between linear transformations?

First, it makes the combination of transformations (linear + activation) nonlinear, thus justifying applying several transformations in a row. This is because the composition of (multivariate) linear functions would still be linear. Therefore, without the activation functions, the network would only be able to approximate combinations of linear functions. For more information, refer to the universal approximation theorem (one of the assumptions is having a non-linearity σ).
Second, it allows choosing an output interval different from R^ny.

4. What are the sizes nx, nh, ny in the figure 1? In practice, how are these sizes chosen?

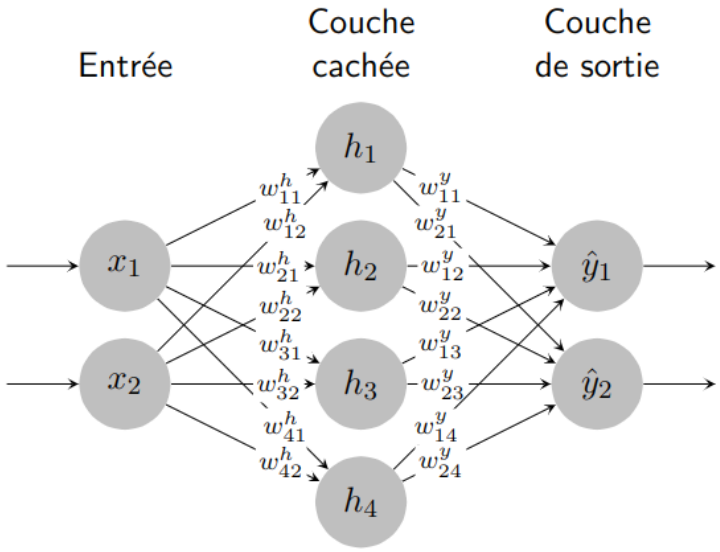


Figure 1: Neural network architecture with only one hidden layer.

In theory, n_x is determined by the size of the input data; therefore, apart from the choice of how to encode the input data, we do not have to choose this value. n_h should be sufficiently large to retain salient information in the process of passing data through the network but sufficiently small for the neural network to extract the most important features. The smaller n_h, the tighter the bottleneck of our neural network. n_y is determined by the purpose of our neural network. For classification, for example, n_y depends on the number of output classes.

Specifically, in figure 1, the size of n_x is 2, n_y is 2, and n_h is 4.

5. What do the vectors ˆy and y represent? What is the difference between these two quantities?

y represents the true class labels (ground truth), and $\hat{y}$ represents the predicted class labels or class probabilities as the output of the network. The model aims to achieve $y = \hat{y}$ as accurately as possible.

6. Why use a SoftMax function as the output activation function?

The softmax function normalizes the output probabilities so that they add up to one. It is utilized for multi-class classification (generally involving more than 2 classes; otherwise, it is preferable to use the sigmoid function). Consequently, the softmax function can be considered a generalization of the sigmoid function, which was employed to represent a probability distribution over a binary variable. Note that we usually avoid using $argmax()$ because it heavily influences the output data, flattening the output vector to one where '1' corresponds to the predicted label and '0' to everything else. Thus, softmax is named as such because it retains more information about the weights: it assigns the most weight to class 1 and less weight to the other classes, thereby retaining the "error."

7. Write the mathematical equations allowing to perform the forward pass of the neural network, i.e. allowing to successively produce $\tilde{h}$, h, $\tilde{y}$ and $\hat{y}$ starting at x.

$$h = \sigma_{hidden}(Wx + b_h) = \mathrm{ReLu}(Wx + b_h)$$

$$\hat{y} = \sigma_{out}(Wh + b_y) = \mathrm{SoftMax}(Wx + b_y)$$

with

$$\mathrm{Softmax}(x)_i := \frac{e^{x_i}}{\sum_k e^{x_k}}$$

Instead of ReLu, we can use any other activation function, such as the $tanh$.

1.3 Loss function

8. During training, we try to minimize the loss function. For cross entropy and squared error, how must the $\hat{y_i}$ vary to decrease the global loss function L?

For cross-entropy loss, you adjust$\hat{y_i}$ to align with the true class probabilities, while for squared error loss, you adjust$\hat{y_i}$ to minimize the squared differences between predictions and true values.

Examining the mathematical formula for cross-entropy, expressed as:

$\mathrm{CE} = l(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$

We can identify two limiting cases:

- If y_i is 1 (indicating the correct label), then $\log(\hat{y_i})$ should be significantly greater than 1. Hence, $\hat{y_i}$ should be greater than 1 to decrease the overall loss (remembering the negative sign).
- If y_i is 0 (indicating the wrong label), $\log(\hat{y_i})$ could be any number; therefore, it's preferable to assign $\hat{y_i}$ as close to 0 as possible. It's important to note that although y_i is 0 and multiplies $\log(\hat{y_i})$, $\hat{y}$ is a vector that should have one label close to 1 and the others close to 0. So, we don't want $\hat{y_i}$ to be selected as 1. Furthermore, in the limit case where $\hat{y_i}$ is 0, in theory, it would result in $0 \times -\infty$. However, because the logarithm function grows slower than the linear function, the function approaches 0 and not $-\infty$. In practice, an arbitrary epsilon constant (e.g., 10^{-10}) is often added to $\hat{y_i}$ to avoid concerns about this limit case.

Similarly, analyzing the mathematical formula for the mean squared error:

$\mathrm{MSE} = ||y - \hat{h}||_2^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$

- If y_i is 1 (indicating the correct label), then$\hat{y_i}$ should be 1.
- If y_i is 0 (indicating the wrong label),$\hat{y_i}$ should be 0. Considering the squaring operation, any negative number will be converted to its absolute value, resulting in an increase in the loss, which is not desired.

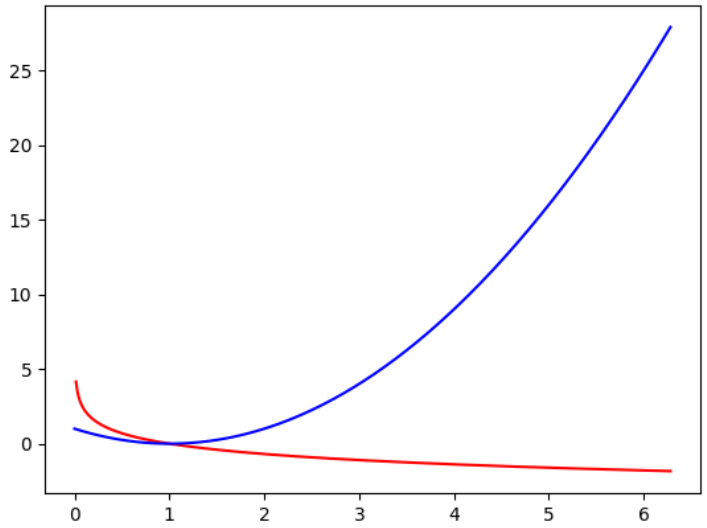
In general:

- If y_i is 1 (indicating the correct label),$\hat{y_i}$ should ideally be as close to 1 as possible.
- If y_i is 0 (indicating the wrong label),$\hat{y_i}$ should ideally be as close to 0 as possible.

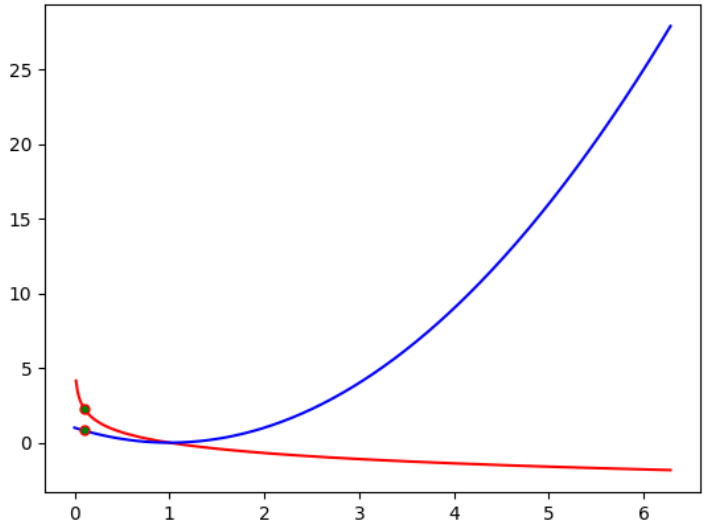
9. How are these functions better suited to classification or regression tasks?

The MSE loss is more suitable for regression tasks (see χ^2 methods).
In classification problems, since we have classes as outputs, we aim for our loss to penalize errors significantly. The MSE doesn't penalize the loss as much as it should. Comparing the two formulas:
$\text{CE} = l(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$
$\text{MSE} = ||y - \hat{h}||_2^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

Let's write the formula in the case of a single element of the y or $\hat{y}$ vector:
$l(y, \hat{y}) = -y * log\hat{y}$
$l(y, \hat{y}) = y - \hat{y}$
For the sake of simplicity, let's consider the case of y=1:
$l(y, \hat{y}) = -log\hat{y}$
$l(y, \hat{y}) = (1 - \hat{y})^2$
If we plot these 2 function we get the following:



If we get $\hat{y} = 0.1$, then we get these points:



Here the CE function is higher then the penalty will be higher.

Let's write the formula in the case of a single element of the y or $\hat{y}$ vector in the case that 1.

1. **Cross-Entropy Loss for y=0**:
   The formula for CE loss for y=0 is: $l(y, \hat{y}) = -y \cdot log(\hat{y})$
   In this case, where y=0: $l(0, \hat{y}) = -0 \cdot log(\hat{y}) = 0$

When the true label y is 0, the CE loss is always 0, regardless of the predicted value $\hat{y}$. This is because the loss only measures the divergence when the true label is 1.

2. **Mean Squared Error Loss for y=0**:
   The formula for MSE loss for y=0 is: $l(y, \hat{y}) = (y - \hat{y})^2$

In this case, where y=0: $l(0, \hat{y}) = (0 - \hat{y})^2 = \hat{y}^2$

When the true label y is 0, the MSE loss is directly proportional to the square of the predicted value $\hat{y}$. This means that the MSE loss will increase as $\hat{y}$ moves away from 0.

So, in contrast to cross-entropy loss, which doesn't penalize the model when the true label is 0, the mean squared error loss will penalize the model by the square of the predicted value. As you mentioned, the choice of the loss function depends on the nature of the problem and the desired behavior. Cross-entropy is better suited for classification problems because it penalizes incorrect class predictions, especially when the true label is 1, while MSE is designed for regression tasks and may not be as suitable for classification problems.

Why don't we use CE for a regression problem? This is because in a regression problem, we are dealing with continuous values. So, even if our regression model gives an answer a little different from the ground truth, it should not cost the model too much.
reference

1.4 Optimization algorithm

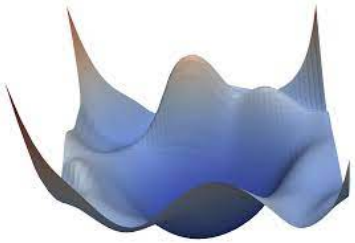
10. What seem to be the advantages and disadvantages of the various variants of gradient descent between the classic, mini-batch stochastic and online stochastic versions? Which one seems the most reasonable to use in the general case?

It computes the exact gradient of the loss function using the entire training dataset in each iteration; therefore, it's more accurate and provides a smoother convergence trajectory. Calculating the entire gradient loss is a very computationally intensive task. That's why it's usually preferred to use mini-batch stochastic gradient descent. In fact, in stochastic gradient descent, we divide the training set into small parts called mini-batches and compute the gradient on them. Since we're computing the gradient descent using a much smaller amount of data, we achieve faster computations than the classical approach.

Mini-batch stochastic gradient descent updates the model's parameters based on one training example at a time, which is computationally very efficient. An example of its use is when dealing with a stream of data. However, since it doesn't provide much data for computation, the gradient descent path during training could be unstable. In general, mini-batch SGD is often considered the most reasonable choice for training neural networks and machine learning models. It combines the advantages of both classic gradient descent and online SGD while mitigating some of their disadvantages.

11. What is the influence of the learning rate η on learning?

The optimization process of the loss function is mathematically uncertain and cannot be solved in an analytical form. This is why we have to rely on numerical methods or optimization schemes that take into account the local gradient with respect to the loss. The "loss landscape" refers to the multidimensional representation of the loss function in relation to various parameters, showing the challenges in finding the optimal solution due to the function's non-convex nature, impacting the search for the global minimum during optimization. This figure shows a loss landscape concerning our input data (the loss function is expressed as: $L(\theta_i|\{x_i\})$. It's not guaranteed that we can find the global minimum because the loss function is not necessarily a log-convex function or a convex function.



The optimization schemes depend on several hyperparameters, including the learning rate, momentum, or Nesterov's acceleration.

The learning rate η determines the incremental rate at which the gradient descent algorithm is updated. A large η will take big steps and converge faster to a sensible point but might overshoot and fail to find finer details (finetuning).

On the contrary, a learning rate that is too small will result in getting stuck in local minima or not converging fast enough. To enhance performance, different update algorithms have been developed that take into account factors such as momentum (Adam) or the future state (Ada).

12. Compare the complexity (depending on the number of layers in the network) of calculating the gradients of the loss with respect to the parameters, using the naive approach and the backpropagation algorithm.

Naive Approach:
In the naive approach, you would directly compute the gradients of the loss with respect to the parameters using the definition of the derivative.

- For each parameter, you perform a forward pass through the entire network, which has a computational cost of $O(N)$ for one parameter.

- You need to compute the derivative with respect to each parameter individually, resulting in $O(N)$ operations for each parameter.

So, the total computational complexity for the naive approach is $O(N^2)$, as you're repeating this process for each parameter. This becomes impractical as the number of parameters increases.

Backpropagation Algorithm:

Backpropagation is a much more efficient and scalable approach for calculating gradients in neural networks. It leverages the chain rule of calculus to compute gradients layer by layer, starting from the output layer and moving backward through the network. The algorithm computes the gradients efficiently without redundant calculations.

- You perform one forward pass through the network, which has a computational cost of $O(N)$ where N is the total number of parameters in the network.
- You then perform a backward pass, which computes gradients layer by layer. The backward pass is roughly $O(L)$ because you compute gradients for each layer in sequence.
- Within each layer, the computation of gradients is $O(N)$, as it depends on the number of parameters in that layer.

The total computational complexity of backpropagation is $O(L * N)$, which scales linearly with the number of layers and parameters in the network. This is significantly more practical for deep neural networks.

13. What criteria must the network architecture meet to allow such an optimization procedure ?

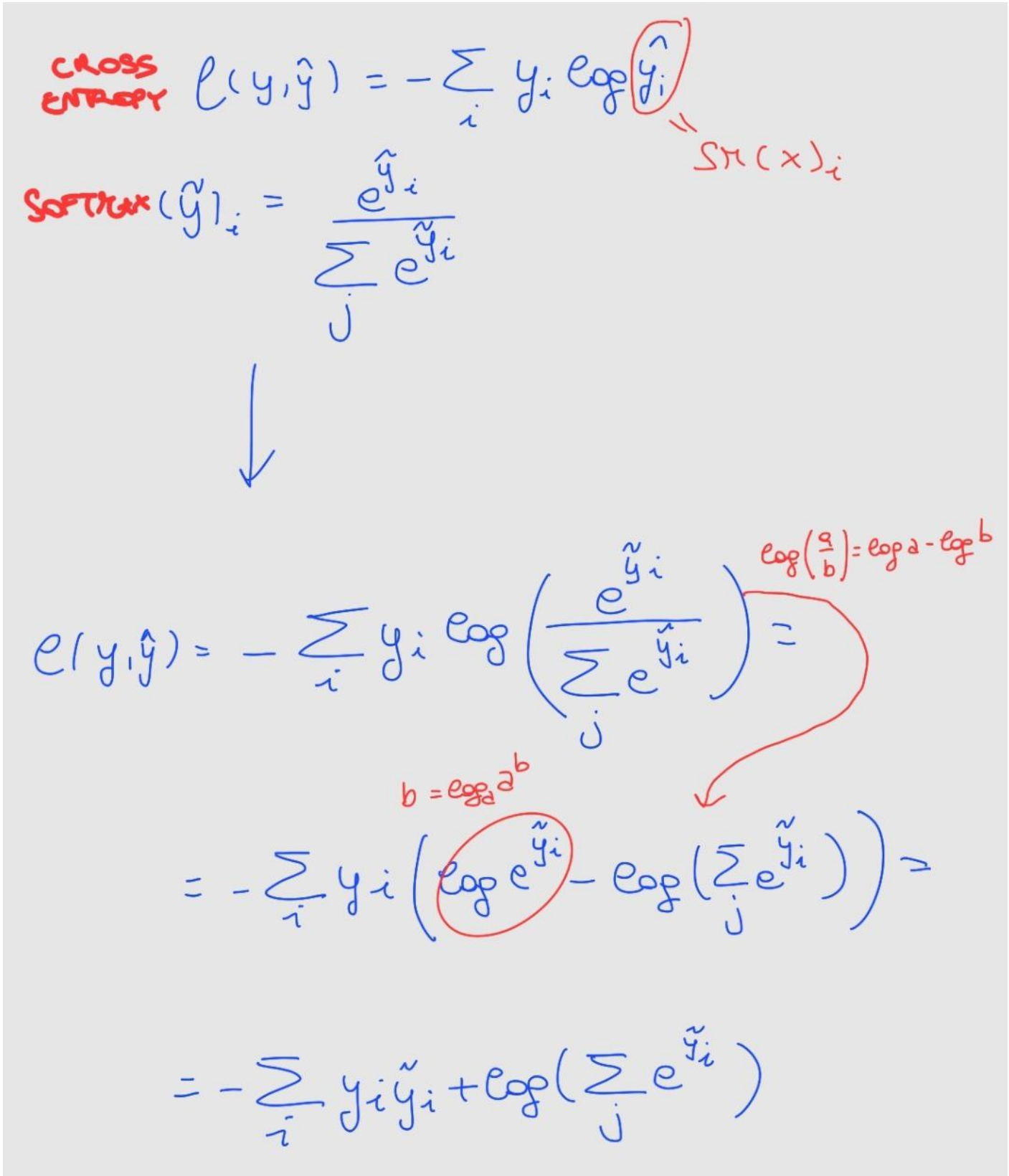
Layer Connectivity: The architecture should have a clear and sequential structure with well-defined layer connections. Each layer should connect to the next layer, and information should flow forward without skipping or looping connections.

Feedforward Structure: Neural networks typically have a feedforward structure where data flows from input layers to output layers without feedback loops. Recurrent neural networks (RNNs) are an exception, but they have a different training procedure.

Differentiability: The network's activation functions and loss function must be differentiable. This is essential for gradient-based optimization methods like backpropagation to calculate gradients and update the network's parameters.

14. The function SoftMax and the loss of cross-entropy are often used together and their gradient is very simple. Show that the loss can be simplified by

$$l = -\sum_i y_i \tilde{y}_i + log(\sum_i \exp(\tilde{y}_i)$$



15. Write the gradient of the loss (cross-entropy) relative to the intermediate output $\tilde{y}$

$$\frac{\partial \ell}{\partial \tilde{y}_i} = \dots \qquad \Rightarrow \qquad \nabla_{\tilde{y}} \ell = \begin{bmatrix} \frac{\partial \ell}{\partial \tilde{y}_1} \\ \vdots \\ \frac{\partial \ell}{\partial \tilde{y}_{n_y}} \end{bmatrix} = \dots$$

$$\frac{\partial e}{\partial \tilde{y}_i} = \sum_{i=1}^{n_y} \left(-y_i + \frac{e^{\tilde{y}_i}}{\sum_{j=1}^{n_{\tilde{y}}} e^{\tilde{y}_j}} \right) =$$

$$= \sum_{i=1}^{n_y} \text{Softmax}(\tilde{y}_i) - y_i$$

$$= \sum_{i=1}^{n_y} \hat{y}_i - y_i$$

16. Using the backpropagation, write the gradient of the loss with respect to the weights of the output layer $\nabla_{W_y} l$. Note that writing this gradient uses $\nabla_{\tilde{y}} l$. Do the same for $\nabla_{b_y} l$.

16)

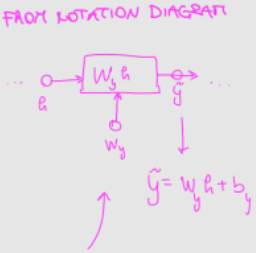
$$\nabla_{W_y} e = \frac{\partial e}{\partial W_{y,i,j}} = \frac{\partial e}{\partial \tilde{y}} \cdot \frac{\partial \tilde{y}}{\partial W_y} =$$

VECTOR CASE

$$= \sum_{k=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_k} \frac{\partial \tilde{y}_k}{\partial W_{y,i,j}} =$$

we know, see answer 15

FROM NOTATION DIAGRAM



$$\frac{\partial \tilde{y}_k}{\partial W_{y,i,j}} = \frac{\partial \left([W_k^y h + b^y] \right)_k}{\partial W_{y,i,j}} = \frac{\partial \tilde{y}_k}{\partial W_{y,i,j}} \left(\sum_{p=1}^{n_h} W_{kp} h_p \right) =$$

$$= \begin{cases} h_j & \text{if } i=k \\ 0 & \text{otherwise} \end{cases}$$

$$\Downarrow \qquad (\hat{y}_i - y_i) \qquad h_j$$

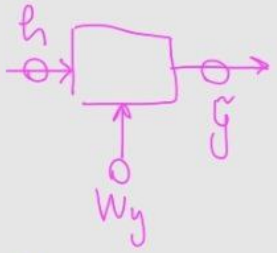
$$\frac{\partial e}{\partial W_{y,i,j}} = \sum_{k=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial W_{y,i,j}} = (\hat{y}_i - y_i) \cdot h_j$$

USING THE NABLA NOTATION :

$$\nabla_{W_y} e = \nabla_{\tilde{y}} e \cdot h$$

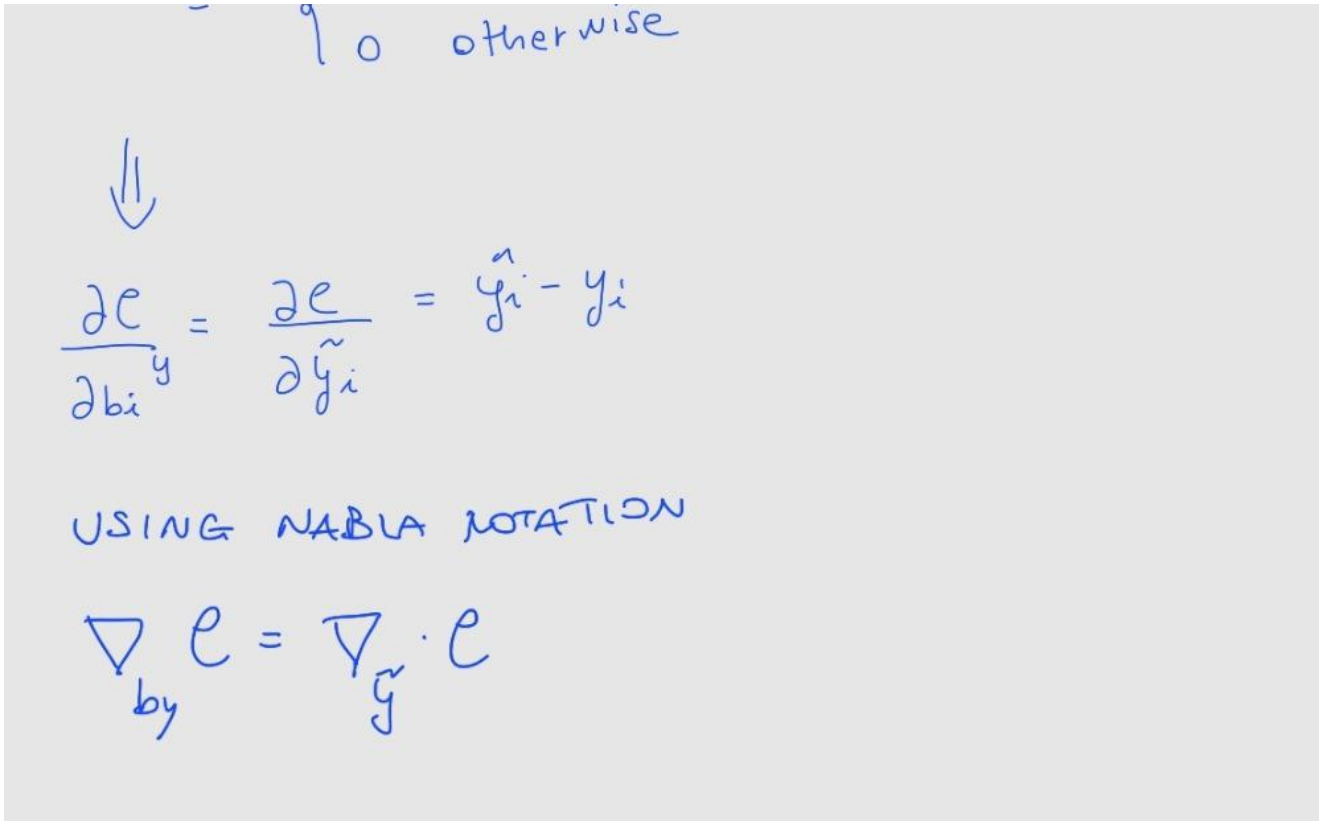
$$\nabla_{b_y} e = \frac{\partial e}{\partial b_i^y} = \sum_{k=1}^{n_y} \frac{\partial e}{\partial \hat{y}_k} \cdot \frac{\partial \tilde{y}_k}{\partial b_i^y}$$

FROM NOTATION DIAGRAM



$$\frac{\partial \tilde{y}_k}{\partial b_i^y} = \frac{\partial \left[W^y \cdot h + b^y \right]_k}{\partial b_i^y} = \frac{\partial \left(b_k^y \right)}{\partial b_i^y} =$$

$$= \begin{cases} 1 & \text{if } i=k \end{cases}$$



17. Compute other gradients : $\nabla_{\tilde{h}} l$**,** $\nabla_{W_h} l$**,** $\nabla_{b_h} l$

17)

$$\nabla_{\tilde{h}} e = \frac{\partial e}{\partial \tilde{h}_i} = \sum_{\kappa} \left(\frac{\partial e}{\partial h_\kappa}\right)^{*} \left(\frac{\partial h_\kappa}{\partial \tilde{h}_i}\right)^{*} =$$

(*)

$$\frac{\partial e}{\partial h_\kappa} = \sum_{q=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_q} \cdot \left(\frac{\partial \tilde{y}_q}{\partial h_\kappa}\right)^{*}$$

SEE ANSWER 15

(*)

$$\frac{\partial \tilde{y}_q}{\partial h_\kappa} = \frac{\partial \left([W^y h + b^y]_q\right)}{\partial h_\kappa} = \frac{\partial [W^y \cdot h]_q}{\partial h_\kappa} =$$

$$= \frac{\partial \left(\sum_{p=1}^{n_h} W^y_{q,p} \cdot h_p\right)}{\partial h_\kappa} = W^y_{q,\kappa}$$

(*) CONSIDERING $h_\kappa = \tanh(\tilde{h}_\kappa)$

$$\frac{\partial h_\kappa}{\partial \tilde{h}_i} = \begin{cases} 1 - \tanh^2(\tilde{h}_i) & \text{if } i = \kappa \\ 0 & \text{otherwise} \end{cases}$$

$$\Downarrow$$

$$\frac{\partial e}{\partial \tilde{h}_i} = \sum_{\kappa=1}^{n_y} \frac{\partial e}{\partial \tilde{y}_q} \cdot \frac{\partial \tilde{y}_q}{\partial h_\kappa} \cdot \frac{\partial h_\kappa}{\partial \tilde{h}_i} =$$

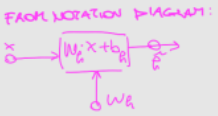
$$= \sum_{i}^{n_y} (\hat{y}_i - y_i) \cdot \sum_{q,\kappa}^{n_h} W^y_{q,\kappa} \cdot (1 - \tanh^2(\tilde{h}_\kappa))$$

USING NABLA NOTATION

$$\nabla_{\tilde{h}_i} e = \nabla_{\tilde{y}} e \cdot W_y \cdot (1 - h^2) \quad \text{with } h = \tanh(\tilde{h})$$

$$\nabla_{W_h} e = \frac{\partial e}{\partial W_h} = \frac{\partial e}{\partial \tilde{h}} \cdot \frac{\partial \tilde{h}}{\partial W_h} =$$

FROM NOTATION PAGRAM:



$$= \sum_{k=1}^{n_h} \frac{\partial e}{\partial \tilde{h}_k} \cdot \frac{\partial \tilde{h}_k}{\partial W_{h,ij}} \quad (*) \qquad \text{where} \quad \begin{matrix} 1 \le i \le n_h \\ 1 \le j \le n_x \end{matrix}$$

$\frac{\partial e}{\partial \tilde{h}_k}$: SEE PREVIOUS ANSWER

$$(*)\ \frac{\partial \tilde{h}_k}{\partial W_{h,ij}} = \frac{\partial \left([W^h x + b^h]_k \right)}{\partial W_h} = \begin{cases} x_j & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

$$\Downarrow$$

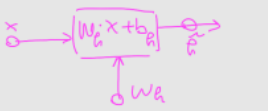
$$\frac{\partial e}{\partial W_h} = \frac{\partial e}{\partial \tilde{h}_i} \cdot x_j = \sum_{j}^{h_k} \sum_{i}^{n_y} (\hat{y}_i - y_i) \cdot \sum_{q,k}^{n_g} W^y_{q,k} \cdot (1 - \tanh^2(\tilde{h}_k)) \cdot x_j$$

USING NABLA NOTATION

$$\nabla_{W_h} e = \nabla_{\tilde{h}} e \cdot x^T = \nabla_{\tilde{y}} e \cdot W_y \cdot (1 - \tilde{h}) \cdot x^T$$

$$\nabla_{b_h} e = \frac{\partial e}{\partial b_h} = \frac{\partial e}{\partial \tilde{h}} \cdot \frac{\partial \tilde{h}}{\partial b_h} =$$

FROM NOTATION DIAGRAM



$$= \sum_{\kappa=1}^{n_{\tilde{h}}} \frac{\partial e}{\partial \tilde{h}_\kappa} \cdot \frac{\partial \tilde{h}_\kappa}{\partial b_{h,i}} \quad \text{where } 1 \le i \le n_h$$

LOOK PREVIOUS ANSWER

$$* \quad \frac{\partial \tilde{h}_\kappa}{\partial b_{h,i}} = \frac{\partial([W^h x + b^h]_\kappa)}{\partial b_h} = \begin{cases} 1 & \text{if } i = \kappa \\ 0 & \text{otherwise} \end{cases}$$

$$\Downarrow$$

$$\frac{\partial e}{\partial W_h} = \sum_{\kappa=1}^{n_{\tilde{h}}} \frac{\partial e}{\partial \tilde{h}_\kappa}$$

USING NABLA NOTATION

$$\nabla_{b_h} e = \nabla_{\tilde{h}} e = \nabla_{\tilde{y}} e \cdot W_j \cdot (1 - h^2) \quad \text{with } h = \tanh(\tilde{h})$$